Genesys Interaction Recording API

Interaction Recording Web Services API

5/15/2026

Contents

Interaction Recording Web Services API

Interaction Recording Web Services provides access to the following APIs:

- Search, Playback, and Delete API — Use this API to search for, play back, and delete recordings stored in the Genesys Interaction Recording (GIR) system.
- Settings API — You'll typically only need this API if you plan to enable recording privacy settings in the Genesys Interaction Recording API.
- Insertion API — Use this API to update the GIR database with records of existing call recordings already stored at a WebDAV storage location, so that they are accessible from the Genesys Interaction Recording user interfaces.
- Call Recording API — Use this API to control call recording functionality in your agent application.
- Recording Label API — Use this API to create and administer label definitions and label recordings.
- Recording Non-Deletion API — Use this API to tag and untag recordings for non-deletion.

The following topics are important for you to review as you'll need this information while using the Interaction Recording Web Services APIs.

Configuring Roles for Recording Users

Interaction Recording Web Services does not use standard Genesys access controls. Instead, it uses its own role-based security depending on settings in the **Annex** tab. The htcc/roles key must be defined in the **Options** of the **Person** object that you use to connect to the API. For example:

```
[htcc]
roles=supervisor
```

Role	Description
agent	Provides agent access. Agents are contact center employees who handle calls, hold chats sessions, and answer emails.
supervisor	Provides supervisor access. A resource whose primary role in the business is direct management of agents and may occasionally engage in the interaction-handling process during coaching or in emergency situations.
admin	Provides administrator access. An employee in the contact center who can create and edit other users, create reason codes, and assign skills to supervisors.
apiuser	Provides the same level of access as an administrator. Use this permission to designate an API user of system account that must be used by

Role	Description
	other server applications and this user does not represent an actual person.

> **Tip**
>
> You can link roles together as comma-separated values. For example:
>
> ```
> [htcc]
> roles=agent,supervisor,admin
> ```

Cross-Site Request Forgery protection

Interaction Recording Web Services provides protection against Cross-Site Request Forgery (CSRF) attacks by requiring a token in a custom header for all requests that modify data: PUT, POST, DELETE. Interaction Recording Web Services generates and stores this token along with the HTTP session. The token shares the life cycle of the HTTP session.

See CSRF protection for details about how to enable this security feature.

To get the CSRF token and the expected header name from Interaction Recording Web Services, just send a GET request — for example, **/api/v2/me**. The expected header name and token value are returned in two custom headers on the HTTP response: `X-CSRF-HEADER` and `X-CSRF-TOKEN`.

```
X-CSRF-HEADER: X-CSRF-TOKEN
X-CSRF-TOKEN: 4a92be65-ec55-4aa2-b9df-9518fd870f2f
```

You must cache the values of these headers because you'll need to use them on subsequent API requests that use PUT, POST, and DELETE so that Interaction Recording Web Services doesn't think the request is coming from a third party. For instance, when you attempt to perform the Insert Recording operation, you need include an HTTP header of X-CSRF-TOKEN with the corresponding value:

```
POST https://htcc-demo.genhtcc.com/internal-api/contact-centers/
57c0b771-b57c-4ea8-8655-7ef6d3c58ccc/recordings HTTP/1.1
Authorization: Basic <credentials>
X-CSRF-TOKEN: 4a92be65-ec55-4aa2-b9df-9518fd870f2f
Accept: application/json, application/xml, text/json, text/x-json, text/javascript,
text/xml
User-Agent: RestSharp/105.2.3.0
Content-Type: application/json
Host: htcc-demo.genhtcc.com
Cookie: JSESSIONID=sngukrzemiyxchpu5isbufmm;
AWSELB=854B09E30CD5CEDDEDA518240935B76DEAC5D82EC5038C4B8F22CD5165FF21C65BC292BAD05CEEB
17D7500F4A489957FB3A5C23BD09BC31CAF09526FCBEFD7CE491CD7E5B3
Content-Length: 88
Accept-Encoding: gzip, deflate
{
   ...
   request body
```

```
   ...
}
```

If you don't have that header in place, Interaction Recording Web Services returns an HTTP 403 error with a response in the Content of "Missing or invalid Csrf token".

Cookie support

In addition to the CSRF feature, Interaction Recording Web Services also requires your application to support cookies, specifically for the cookie that it returns. Without a cookie store, Interaction Recording Web Services returns the same HTTP 403 error with a message of "Missing or invalid Csrf token", even if the X-CSRF-TOKEN is specified in the HTTP Header. This is because it can't confirm that the X-CSRF-TOKEN you specify lines up with the cookie that the token is supposed to be tied to.

Read on for some sample requests and examples of how to implement CSRF protection:

Authorized request returning token headers

Request

```
GET /api/v2/me

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/35.0.1916.153 Safari/537.36
Authorization: Basic cGF2ZWxkQHJlZHdpbmdzLmNvbTpwYXNzd29yZA==
Accept: */*
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=hac082exio454jcqk6ieqm4j
```

Response

```
200 - OK
Date: Mon, 23 Jun 2014 02:00:15 GMT
X-CSRF-HEADER: X-CSRF-TOKEN
Set-Cookie: JSESSIONID=1h49t997p4mgc1e108bz0cjntr;Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
X-CSRF-TOKEN: e2fcfafd-c600-4156-88ae-ca56babd24e1
Pragma: no-cache
Cache-Control: no-cache
Cache-Control: no-store
Content-Type: application/json
Transfer-Encoding: chunked
```

POST request including CSRF token

Request

```
POST /api/v2/me

Origin: chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo
X-CSRF-TOKEN: e2fcfafd-c600-4156-88ae-ca56babd24e1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/35.0.1916.153 Safari/537.36
Content-Type: application/json
Accept: */*
```

```
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=1h49t997p4mgc1e108bz0cjntr

{ "operationName": "Ready" }
```

Response

```
200 - OK
Date: Mon, 23 Jun 2014 02:02:51 GMT
Pragma: no-cache
Cache-Control: no-cache
Cache-Control: no-store
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(8.1.14.v20131031)

{ "statusCode": 0 }
```

JavaScript example

```
<html>
        <head>

                <script type="text/javascript" src="./org/cometd.js"></script>
            <script type="text/javascript" src="./org/cometd/ReloadExtension.js"></script>
                <script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/
jquery.min.js"></script>
                <script src="./jquery.cometd.js"></script>

                <script>

                ////////////////////////////////////////////////////////////////////////
                // Initialization
                ////////////////////////////////////////////////////////////////////////
                var baseUri = 'http://127.0.0.1:8080';
                var username = 'paveld@redwings.com';
                var password = 'password';

                var csrfHeaderName;
                var csrfToken;
                var cometd;

                $.ajaxSetup({
             beforeSend: function(xhr) {
                    if (csrfHeaderName && csrfToken) {
                       xhr.setRequestHeader(csrfHeaderName, csrfToken);
                    }
             }
         });

                $(document).ready(function() {
                        $('#getMeButton').click(getMe);
                        $('#startCometdButton').click(connectCometD);
                        $('#startSessionButton').click(startContactCenterSession);
                        $('#readyButton').click(ready);
                        $('#stopCometdButton').click(disconnectCometD);
                        $('#endSessionButton').click(endContactCenterSession);

                        cometd = $.cometd;
```

```
                    cometd.addListener('/meta/handshake', onHandshake);
                cometd.addListener('/meta/connect', onConnect);
                    cometd.addListener('/meta/disconnect', onDisconnect)

                    $(window).unload(function() {
                            cometd.disconnect();
                    });
            });

            ////////////////////////////////////////////////////////////////////////////
            // HTTP Helpers
            ////////////////////////////////////////////////////////////////////////////
            var get = function(params)
            {
                    var request = {
                url: baseUri + params.uri,
                type: 'GET',
                            crossDomain: true,
                            xhrFields: {
                                withCredentials: true
                            },
                success: function (data, textStatus, response) {
                        console.log(response.getAllResponseHeaders());

                        if (response.getResponseHeader('X-CSRF-HEADER') &&
response.getResponseHeader('X-CSRF-TOKEN')) {
                            csrfHeaderName = response.getResponseHeader('X-CSRF-HEADER');
                            csrfToken = response.getResponseHeader('X-CSRF-TOKEN');

                            console.log('csrfHeaderName: ' + csrfHeaderName);
                            console.log('csrfToken: ' + csrfToken);
                    }

                    if (params.callback) {
                                params.callback(data);
                        }
                },
                error: function (result) {
                    console.log(result);

                    if (params.error) {
                                            params.error(result);
                                    }
                }
            };

            if (params.includeCredentials) {
                    request.beforeSend = function (xhr) {
                                    xhr.setRequestHeader('Authorization', 'Basic ' +
window.btoa(username + ':' + password));
                            };
                    }

            $.ajax(request);
            };

            var post = function(params)
            {
                    var data = JSON.stringify(params.json, undefined, 2);

            var request = {
                url: baseUri + params.uri,
                type: 'POST',
```

```
                data: data,
    headers: {
        'Content-Type' : 'application/json'
    },
                crossDomain: true,
                xhrFields: {
                    withCredentials: true
                },
    handleAs: 'json',
    success: function(data) {
            if (params.callback) {
                    params.callback(data);
            }
    },
    error: function (req, err, exception) {
        console.log('Error! (' + req.status + ') : ' + err + ' ' + exception);
        if (params.error) {
                                params.error(result);
                        }
    }
       };

$.ajax(request);
}

////////////////////////////////////////////////////////////////////////////
// API Functions
////////////////////////////////////////////////////////////////////////////
var getMe = function() {
        get({
                uri: '/api/v2/me',
                includeCredentials: true
        });
};

var startContactCenterSession = function() {
        post({
                uri: '/api/v2/me',
                json: {
                        operationName: 'StartContactCenterSession',
                        channels: ['voice']
                }
        });
};

var ready = function() {
        post({
                uri: '/api/v2/me',
                json: {
                        operationName: 'Ready'
                }
        });
};

var endContactCenterSession = function() {
        post({
                uri: '/api/v2/me',
                json: {
                        operationName: 'EndContactCenterSession'
                },
                callback: onEndContactCenterSessionComplete
        });
};
```

```
		////////////////////////////////////////////////////////////////////////////
		// Callbacks
		////////////////////////////////////////////////////////////////////////////
		var onEndContactCenterSessionComplete = function() {
			csrfHeaderName = null;
			csrfToken = null;
		}

		////////////////////////////////////////////////////////////////////////////
		// CometD
		////////////////////////////////////////////////////////////////////////////

		var connected = false;
		var subscription;

		var onConnect = function(message) {
			if (cometd.isDisconnected()) {
				return;
			}

			var wasConnected = connected;
			connected = message.successful;
			if (!wasConnected && connected) {
				console.log('Cometd connected.');
			} else if (wasConnected && !connected) {
				console.log('Cometd disconnected...');
			}
		};

		var onDisconnect = function(message) {
			if (message.successful) {
				connected = false;
				console.log('Cometd disconnected.');
			}
		};

		var onMessage = function(message) {
			console.log('Cmetd message received:\n' + JSON.stringify(message,
null, 2));
		};

		var onHandshake = function(handshake) {
		if (handshake.successful === true) {
				if (subscription) {
					console.log('unsubscribing: ' + subscription);
					cometd.unsubscribe(subscription);
				}

				console.log('Subscribing to channels...');
				subscription = cometd.subscribe('/v2/me/*', onMessage);
		}
		};

		var connectCometD = function() {

			var reqHeaders = {};
			reqHeaders[csrfHeaderName] = csrfToken;

			cometd.unregisterTransport('websocket');
			cometd.unregisterTransport('callback-polling');
			cometd.configure({
			    url: baseUri + '/api/v2/notifications',
```

```
                            logLevel: "info",
                            requestHeaders: reqHeaders
                        });

                        cometd.handshake();
                };

                var disconnectCometD = function() {
                        cometd.disconnect();
                };

                </script>
        </head>

        <body>
                <button id='getMeButton'>Get Me</button>
                <br/>
                <button id='startCometdButton'>Start CometD</button>
                </br>
                <button id='startSessionButton'>Start Contact Center Session</button>
                <br/>
                <button id='readyButton'>Ready</button>
                <br/>
                <button id='stopCometdButton'>Stop CometD</button>
                </br>
                <button id='endSessionButton'>End Contact Center Session</button>
        </body>
</html>
```

Python example

```
import base64;
import httplib2;
import json;

GWS_BASE_URI = "http://127.0.0.1:8080/api/v2"
ADMIN_USERNAME = "mikeb@redwings.com"
ADMIN_PASSWORD = "password"

CONTACT_CENTER_USERS = [
    {
        "userName": "bobp@redwings.com",
        "firstName": "Bob",
        "lastName": "Probert",
        "password": "password",
        "phoneNumber": "5019",
        "role": "ROLE_AGENT"
    }
]

X_CSRF_HEADER = "x-csrf-header"
X_CSRF_TOKEN = "x-csrf-token"

jsessionid = None
csrfHeaderName = None
csrfTokenValue = None

http = httplib2.Http(".cache")

def create_request_headers():
    request_headers = dict()
    request_headers["Content-Type"] = "application/json"
```

```
    request_headers["Authorization"] = "Basic " + base64.b64encode(ADMIN_USERNAME + ":" +
ADMIN_PASSWORD)

    if jsessionid:
        request_headers["Cookie"] = jsessionid;
        print "Using JSESSIONID %s" % jsessionid;

    if csrfHeaderName and csrfTokenValue:
        print "Adding csrf header [%s] with value [%s]..." % (csrfHeaderName, csrfTokenValue)
        print
        request_headers[csrfHeaderName] = csrfTokenValue
    else:
        print "No csrf token, skipping..."
        print

    return request_headers

def post(uri, content):
    request_headers = create_request_headers()
    body = json.dumps(content, sort_keys=True, indent=4)

    print "POST %s (%s/%s)..." % (uri, ADMIN_USERNAME, ADMIN_PASSWORD)
    print body
    print

    response_headers, response_content = http.request(uri, "POST", body = body, headers =
request_headers)
    status = response_headers["status"]

    ugly_response = json.loads(response_content)
    pretty_response = json.dumps(ugly_response, sort_keys=True, indent=4)

    print "Response: %s" % (status)
    print "%s" % (pretty_response)
    print

    return response_headers, ugly_response

def get(uri):

    global csrfHeaderName
    global csrfTokenValue
    global jsessionid

    request_headers = create_request_headers()
    print "GET %s (%s/%s)..." % (uri, ADMIN_USERNAME, ADMIN_PASSWORD)
    print

    response_headers, response_content = http.request(uri, "GET", headers = request_headers)
    status = response_headers["status"]
    if response_headers["set-cookie"]:
        jsessionid = response_headers["set-cookie"]
        print "Set JSESSIONID %s..." % jsessionid

    ugly_response = json.loads(response_content)
    pretty_response = json.dumps(ugly_response, sort_keys=True, indent=4)

    print "Response: %s" % (status)
    print "%s" % (pretty_response)
    print

    if X_CSRF_HEADER in response_headers:
        csrfHeaderName = response_headers[X_CSRF_HEADER]
```

```
        print "Saved csrf header name [%s]" % csrfHeaderName

    if X_CSRF_TOKEN in response_headers:
        csrfTokenValue = response_headers[X_CSRF_TOKEN]
        print "Saved csrf token value [%s]" % csrfTokenValue
        print

    return response_headers, ugly_response

def check_response(response_headers, expected_code):
    if response_headers["status"] != expected_code:
        print "Request failed."
        exit(-1)

def create_user(user_info):
    user_name = user_info["userName"]
    print "Creating user [%s]..." % (user_name)

    uri = "%s/users" % (GWS_BASE_URI)

    user = {
        "userName": user_name,
        "password": user_info["password"],
        "firstName": user_info["firstName"],
        "lastName": user_info["lastName"],
        "roles": [user_info["role"]]
    }

    response_headers, response_content = post(uri, user)
    check_response(response_headers, "200")

    user_id = response_content["id"]
    print "User [%s] created. User id [%s]." % (user_name, user_id)

    return user_id

def assign_device_to_user(user_id, phone_number):
    print "Creating device [%s] and assigning to user [%s]..." % (phone_number, user_id)

    uri = "%s/users/%s/devices" % (GWS_BASE_URI, user_id)

    device = {
        "phoneNumber": phone_number
    }

    response_headers, response_content = post(uri, device)
    check_response(response_headers, "200")

    device_id = response_content["id"]
    print "Device [%s] created and assigned to user id [%s]." % (device_id, user_id)

def create_users_and_devices():
    for user_info in CONTACT_CENTER_USERS:
        user_id = create_user(user_info)
        assign_device_to_user(user_id, user_info["phoneNumber"])

def getToken():
    uri = "%s/diagnostics/version" % (GWS_BASE_URI)

    response_headers, response_content = get(uri)
    check_response(response_headers, "200")

if __name__ == "__main__":
```

```
getToken()
create_users_and_devices()
```

Cross-Origin Resource Sharing

Cross-Origin Resource Sharing (CORS) is a specification that enables open access across domain-boundaries.

Each contact center can define their own allow origin list through Interaction Recording Web Services access control settings.

Interaction Recording Web Services filters an incoming request by merging global `allowOrigins` and contact center access control settings by using an `Admin` account.

Operations

The following operations are available for this group:

Operation	Description	Permissions
GET	Retrieves an array of settings	Contact Center Admin
POST	Creates a new setting in this group. `allowedOrigins` is the only valid setting.	Contact Center Admin
PUT	Updates a setting.	Contact Center Admin
DELETE	Removes a setting.	Contact Center Admin

Settings

Attribute name	Description
allowedOrigins	An array of valid "origins" for this contact center. The CORS filter will use this list to validate incoming requests.

> **Tip**
> Wildcards are allowed in the context of a domain name for `allowedOrigins`, but `"*"` by itself is not permitted.

Examples

Retrieve access control settings

```
GET /settings/access-control {
   settings:[
{
  “name”:”allowedOrigins”,
  “value”: ["https://cloud.genhtcc.com", "https://*.genhtcc.com"]
}
]
```

Add "genesys.com" to the list of domains

```
PUT /settings/access-control {
   settings:
{
  "name":"allowedOrigins",
  "value": ["https://cloud.genhtcc.com", "https://*.genhtcc.com", "https://*.genesys.com"]
}
```

> **Important**
>
> When sending the above, the entire array must be sent

Return Values

Interaction Recording Web Services API requests always return the `statusCode` attribute. If an error occurs, that is, if the `statusCode` is not `0`, the response includes error details in the `statusMessage` attribute.

Interaction Recording Web Services supports the following status codes:

Code	Description
0	The operation is successful. No `statusMessage` is provided.
1	A required parameter is missing in the request.
2	A specified parameter is not valid for the current state.
3	The operation is forbidden.
4	An internal error occurred. This could occur if an internal error occurred with Interaction Recording Web Services or with one of the servers working with Interaction Recording Web Services (for example: Cassandra or a Genesys Framework component).

Code	Description
5	The user does not have permission to perform this operation.
6	The requested resource could not be found.
7	The operation was partially successful. Returned if at least one action in a bulk operation succeeded. More information is available in the Partial success section.
8	Change password demanded. Interaction Recording Web Services requested a password change for the user.
9	Processing incomplete.
10	Input validation error - the provided value is not within the range of valid values.
11	User requested to change read-only property.
12	Unable to retrieve resource error.
13	Unable to create resource error.
14	Unable to delete resource error.
15	Unable to update resource error.
16	Unable to assign resource error.
17	Unable to unassign resource error.
18	Resource already exists.
19	Resource already in use.
20	User is not authenticated. Any subsequent request should provide credentials.

If an error occurs during an operation, the response includes `statusCode` and `statusMessage` to clarify the error. No other attributes are included.

Note that if an error occurs during a request, you can assume that the request failed to modify the data for the contact center.

Partial success

Some operations may be considered successful if they are able to perform some of their work. These operations are considered "bulk" operations and are different from "transactions", which involve multiple steps that possibly use multiple servers. An example of a transaction is "create user" which involves creating some data in Cassandra as well as Configuration Server. If one of these actions fails, Web Services considers the whole operation a failure. In contrast, an operation such as "assign multiple skills to user" is a bulk operation which consists of a series of transactions (for example, each individual skill assignment is a transaction). The general rule is that if a step of a transaction fails, Web Services considers the whole operation a failure. If at least one transaction in a bulk operation succeeds, Web Services considers this a "partial success." Note that for bulk GETs (for example, GET /users) if the result is a partial list, the response includes statusCode:7 instead of 0. The rest of the result looks the same. For POST, PUT, and DELETE, the partial success returns have the following attributes:

Attribute	Value
statusCode	Always 7
succeeded	An array of resource descriptors (see below). Each represents a resource for which the transaction was successful.
failed	An array of failure descriptors (see below). Each represents a resource for which the transaction failed.

Attribute	Value
uri	The URI of a resource from request parameters for which the transaction succeeded. For example, if assigning multiple skills to a user, this is the URI of a skill).
path	The relative path of the resource
id	The unique identifier of the resource above.

Attribute	Value
<uids>	The attributes which uniquely identify the resource for which this transaction failed. For example, if assigning skill uris, this will be "uri." If creating a user this will be "userName." If a resource has more than one identifying attribute all should be present.
statusCode	The status code describing the reason for failure.
statusMessage	The message describing the reason for failure.

Examples

Assign:

```
POST /users/{id}/skills
{
        "uris":["uri1", "uri2"], "paths": ["uri3"]
}

{
        "statusCode":7 (partial success)
        "succeeded":[
                {
                        "id":<id1>,
                        "uri":"uri1",
                        "path":"path1",
                },
                {
                        "id":<id2>,
                        "uri":"uri2",
                        "path":"path2"
                }]
        "failed":[
                {
                        "statusCode":X,
                        "statusMessage":"msg",
                        "uri":"uri3",
```

```
                        "path": "path2"
                }
        ]
}
```

Create:

```
POST /users
{
        "users":[
                {
                        "firstName":"..",
                        "lastName":"..",
                        "userName":"u1", etc
                },
                {
                        "firstName":"..",
                        "lastName":"..",
                        "userName":"u2", etc
                },
                {
                        "firstName":"..",
                        "lastName":"..",
                        "userName":"u3", etc
                }

        ]
}

{
        "statusCode":7 (partial success)
        "succeeded":[
                {
                        "id":<id>,
                        "uri":"uri1",
                        "path":"path"
                },
                {
                        "id":<id>,
                        "uri":"uri2",
                        "path":"path2"
                }]

        "failed":[
                {
                        "statusCode":3,
                        "statusMessage":"Operation forbidden, username already exists",
                        "userName":"u3"
                }]
}
```

Delete:

```
DELETE /users
{
        "uris":["uri1", "uri2"],"paths": ["uri3"]
}

{
        "statusCode":7 (partial success)
        "succeeded":[
                {
```

```
                        "id":<id1>,
                        "uri":"uri1",
                        "path":"path1"
                },
                {
                        "id":<id2>,
                        "uri":"uri2",
                        "path": "path2"
                }]

        "failed":[
                {
                        "statusCode":X,
                        "statusMessage":"...",
                        "uri":"uri3",
                        "path": "path2"
                }]
}
```